Experimental AI in Games: Papers from the AIIDE Workshop
AAAI Technical Report WS-16-22

Learning Player Tailored Content From Observation: Platformer Level Generation from Video Traces Using LSTMs

Adam Summerville,[1] Matthew Guzdial,[2] Michael Mateas,[1] and **Mark O. Riedl[2]**
[1]Center for Games and Playable Media, University of California, Santa Cruz
[2]School of Interactive Computing, Georgia Institute of Technology
asummerv@ucsc.edu, mguzdial3@gatech.edu, michaelm@soe.ucsc.edu, riedl@cc.gatech.edu

Abstract

A touted use of Procedural Content Generation is generating content tailored to specific players. Previous work has relied on human identification of player profile features which are then mapped to level generator features. We present a machine-learned technique to train generators on Super Mario Bros. videos, generating levels based on latent play styles learned from the video. We evaluate the generators in comparison to the original levels and a machine-learned generator trained using simulated players.

Introduction

Procedural Content Generation (PCG) for video game levels represents a means of creating content tailored to players, generating content to better match a player's style of play. Previous work has focused on a two-step process of:

1. Perform player modeling - Either by collecting the player's subjective experience (via a post play survey), recording physiological response during/after play, or by recording play-trace data (Yannakakis et al. 2013)
2. Generate content based on **(1)** - Either using the player model to set generator parameters (e.g. for the physiological and trace based methods) or as an objective function (typically a regression based on the subjective evaluation)

The process of collecting player data represents a time bottleneck. With the advent of Youtube and players uploading *longplays*, videos of a player's entire playthrough of a game, a large number of play-traces are available for analysis. These do not come with subjective analysis but might come with physiological response (voice recordings might be included as might a facial video recording), lending themselves to certain analysis. In addition, a player's interaction with a space is an implicit encoding of the content they wish to encounter. A *Super Mario* player that collects every coin and power-up will have a different play-trace from a player that speedruns through a level. Figure 1 shows an example of how play style could affect the learning (and therefore generation) process. Given a flat segment both players simply run to the right, but in the presence of blocks (**Segment 2**)

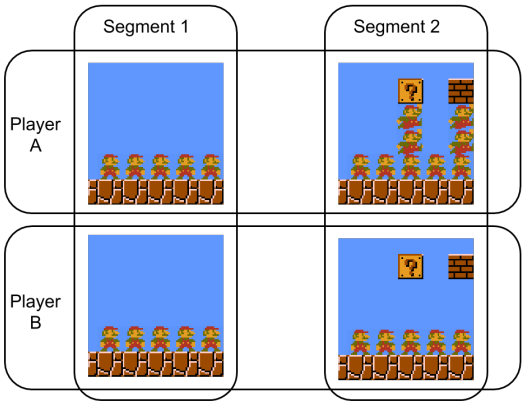

Figure 1: Example of how play-style can influence player/level co-learning and therefore co-generation.

Player A makes sure to jump and hit both while **Player B** continues to run along the ground.

In this work we use Long Short-Term Memory Recurrent Neural Networks (LSTM RNNs) to learn from play-traces gathered from Youtube longplay videos of *Super Mario Bros.*. The LSTM RNN then co-generates levels and predicted paths through said levels according to its learned latent player model. By incorporating a player's path through the level, generated levels are more likely to be completeable as the system learns to predict and generate paths, and more likely to generate tailored content. A player's interactions with a level will bias the generator to create the type of interactions it was trained on. For instance, a generator trained on the traces of a player that jumps and hits every single ?-block will be more likely to include ?-blocks as they are a more integral part of that player's traces. A player that runs under all ?-blocks will be less likely to generate ?-blocks due to the fact that it doesn't matter what a player is under if they are just running along the ground. This comes from the ability of LSTMs to generalize well over sequences, where the sequence in this case is a sequence of player movement and co-occurring level geometry. The co-generation of levels and paths means that the generator is holistic and will generate levels and paths in communication with each other.

Here we present a novel machine-learning-based level

generator architecture that learns to generate levels that incorporate a player's play-style from human play-traces derived from video. Our contributions include (1) a low-effort process for extracting player paths from video, (2) which are used in an experiment demonstrating that LSTM RNN can generate content influenced by latent player style. In the following sections we will discuss related work, followed by a discussion of the data source, our machine learning methodology, our results and some exemplar levels, and finally conclude and discuss future work.

Related Work

Procedural content generation refers to the construction and use of systems capable of generating content, typically for a video game (Hendrikx et al. 2013). The field of *player modeling* focuses on the construction of predictive models of video game player behavior or preference (Yannakakis and Togelius 2011). This work rests between the two fields, interested in biasing a procedural content generator with a particular learned player model. Prior research within this combined space has been referred to as *experience-driven procedural content generation* (Yannakakis et al. 2013).

There has been a great deal of work in procedural content generation of Super Mario Bros. levels (Shaker et al. 2011; Horn et al. 2014). However, the majority of this work relies on human-authored knowledge in the form of rule sets, grammars, and heuristics. Recent data-driven approaches instead extract models of design knowledge from secondary human sources including level maps and gameplay video (Summerville and Mateas 2016; Snodgrass and Ontañón 2014; Dahlskog and Togelius 2014; Guzdial and Riedl 2016). Our work is the first to utilize the benefits of both of these data sources to inform a generator.

Experience-driven procedural content generation has been used in a variety of games, including shooters, racing games, and even Super Mario Bros. (Togelius, De Nardi, and Lucas 2007; Hastings, Guha, and Stanley 2009; Shaker, Shaker, and Abou-Zleikha 2015; Yannakakis and Togelius 2011). In such systems, the human authors define a model of player behavior and how this model impacts the generation of content. For example in Galactic Arms Race (Hastings, Guha, and Stanley 2009), how frequently the player shoots a gun informs the heuristic for the generation of new guns. Our approach eschews an authored, explicit connection between model and generator. Instead, our generator learns level design and player behavior at the same time, and then generates a level with an "intended" player path.

To the best of our knowledge there has been no prior work in the use of gameplay video in experience-driven procedural content generation. However, there exists some work in the use of gameplay videos for player modeling. Barbosa et al. (2014) make use of a grammar of events to extract a "log" of player behavior from gameplay video of Super Mario World, but do not develop a formal player model. Hsieh and Sun (2008) build player "strategy models" from analyzing gameplay video by hand, but do not use these models to generate content. Guzdial and Riedl (2016) used video to inform a data-driven procedural content generator, but had no model of individual players.

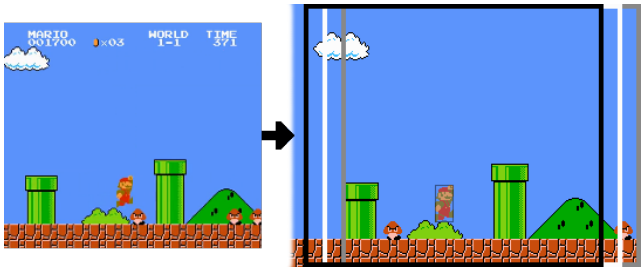

Figure 2: Visualization of the video parsing process.

While there has been significant work in using neural network architectures to play games (Stanley, Bryant, and Miikkulainen 2005; Togelius, De Nardi, and Lucas 2007; Mnih et al. 2015), there has been relatively little work in using them for content generation (Hoover et al. 2015; Hoover, Togelius, and Yannakis 2015). Hoover et al. utilized a neural network architecture to predict parameters for Super Mario Bros. level features. Jain et al. (2016) made use of autoencoders to generate low-fidelity Super Mario Bros. levels. Summerville and Mateas (2016) generated Super Mario Bros. levels with an approach similar to our own, which we will discuss further on. To the best of our knowledge this is the first time neural networks have been used for generating levels to fit a learned play style.

Play Trace Extraction

In order to train our generator, We want to extract a *play trace*, a record of the player's movement through the game world, from each video. Play traces can take a variety of forms, including game events such as player input or enemy deaths. For the purposes of this work we chose to record only spatial information, the player's path recorded as a set of sequential states. We made this choice to minimize the impact of any assumptions in terms of the relative importance of various game events. We represent play traces on maps of the original game levels to better compare between videos and to correct for noise in gameplay video.

Figure 2 visualizes the extraction of player paths. Each frame of video is processed with OpenCV (Pulli et al. 2012), an open machine vision toolkit, to determine its level contents in a process similar to (Guzdial and Riedl 2016). Similarly, OpenCV processes the map images for each game level (such as the segment of game level on the right of Figure 2). The map is then broken into a set of *potential frames*, representing potential placements of the in-game camera, shown as black, white, and gray outlines in the figure. Each *potential frame* has the same width of sixteen in-game tiles, which is the same width as the in-game camera's view width.

After the pre-processing, the system considers each frame with a "player avatar" within it, mapping it to the closest-matching local potential frame and calculating the global position of the player based on this mapping. As seen in Figure 2, this process is imperfect, but sufficient for our purposes. By "closest-matching", we indicate the use of a distance function that transforms both frames and potential frames to a 16x14 matrix of tile ids and then undergoes matrix sub-

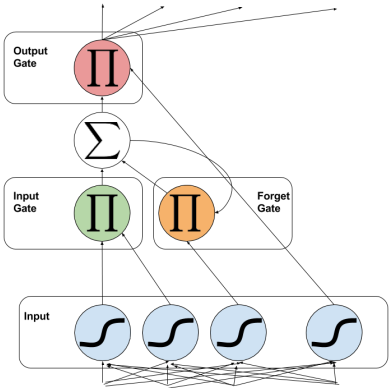

Figure 3: Visualization of an LSTM cell. Reprinted from (Summerville and Mateas 2016)

traction. The set of these tile ids is taken from (Summerville and Mateas 2016). We search locally according to the prior best potential frame in order to avoid noise in the video causing the player to "teleport". The local search is fixed to the eight potential frames before and after the prior best match, allowing for up to a camera-width in movement. The end result of this process is a sequence of player positions on the map such as Figure 1.

The current system has two additional requirements. First, it requires annotated start and end times of level segments in the video. This is required as the current generator does not include "underwater" or "castle" levels. Second, the system requires the start and end time for the entrance and exit of bonus rooms, along with their location on the map, as it does not currently handle the generation of bonus rooms. We look to address both of these assumptions in future work, in order to make the play trace extraction more general.

Method

This work builds on the prior work of Summerville and Mateas, using LSTM RNNs to generate levels. We present a brief overview of their approach in this section, but see (Summerville and Mateas 2016) for greater detail. LSTMs represent the current state of the art of sequence learning. LSTMs were first put forth by Hochreiter and Schmidhuber (Hochreiter and Schmidhuber 1997), but we use the modern formulation with forget gates from Gers et al. (Gers, Schmidhuber, and Cummins 2000), visualized in figure 3. Our neural network architecture makes use of many of these cells, trained on transitions of tiles as we will explain further below. The individual LSTM cells are densely connected to all possible input nodes (the bottom light blue-layer). As input comes in to a given cell it passes through the green $\prod$ which acts as the input gate, determining whether the input makes it into the recurrent connection. If input makes it into the recurrent connection, it is held additively in the white $\sum$ node. The orange $\prod$ node in the second layer represents the forget gate, which allows the recurrent node to 'forget' its retained input. Finally, there is the output gate at the red $\prod$ node at the top. This gate determines whether the cell outputs information to further layers. Taken together, LSTM cells can identify important input, even if it occurs very rarely due to its selective "memory" and "forgetting".

The LSTM RNN of Summerville and Mateas produced generated levels that were able to be completed by an AI (and thus were "well-formed" levels) at a rate greater than the best human-authored rule-based system (97% vs 94% (Summerville and Mateas 2016)), and as such forms the baseline that we wish to compare against. To try to ensure that this comparison is as complete as possible, we use their data formulation and network topology as our basis. The chosen topology is an architecture of 3 densely connected layers that each contain 512 LSTM cells. The output of these layers is fed to a Softmax layer that computes a categorical probability distribution over tile types (e.g. "solid", "enemy", etc). Given a sequence of 200 of these tiles types (˜12 columns of history) it will predict the next tile type.

Super Mario Bros. levels can be represented as a two-dimensional grid of tiles, but the LSTM requires transitions in a linear sequence. There are many potential ways to go from a two to one dimensional representations, but Summerville and Mateas report that the highest performing data formulation is what they labeled **Snaking-Path-Depth**, which we use for this work.

Snaking refers to the path that the sequence takes through the level. The sequence of tiles is derived by going column by column, alternating up and down on each column.

Depth refers to meta-information that is included as part of the input. To try to implicitly encode the fact that levels tend to have different structures at different points, they included 1 special character for every 10 columns into the level (i.e. columns 0-9 had no character, 10-19 had 1, etc.0).

Path refers to the path information encoded in the level input. A key innovation of Summerville and Mateas, this allows the generator to learn to generate not just levels but also plausible player paths within the level. To generate these paths, they used simulated tile-level A* agents. We differ from them here in the use of real player path data, rather than path data from simulated agents.

By incorporating actual human paths extracted from the previously mentioned YouTube longplays, our LSTM RNN is able to learn not just the physics and dynamics that the A* agents afford, but also individual player style. Players will stop, stutter, take side-diversions to collect coins or power-ups, or go out of their way to defeat enemies. All of these are missteps or mistakes for an agent attempting to play optimally, but represent the way the majority of humans approach platformer levels.

We chose to use the Summerville and Mateas tile types to better compare to their generated levels. The tile types are Solid, Enemy, Destructible Block, Question Mark Block With Coin, Question Mark Block With Power-up, Coin, Bullet Bill Shooter Top, Bullet Bill Shooter Column, Left Pipe, Right Pipe, Top Left Pipe, Top Right Pipe, and Empty. It should be noted that certain interactive sprites are categorized in ways that might result in a misunderstanding of the dynamics of the level, as Summerville and Mateas chose to ignore springs and moving platforms. Future work is required to better encode levels to fully capture all of the dy-

	Avg. Time	Distance Ratio	Bonus %
Video A	101.7	2.87	93.3%
Video B	61.8	2.88	83.3%
Video C	48.1	2.57	0%
Video D	48.1	3.14	35%%

Table 1: Descriptive metrics for each video.

namics.

Evaluation

For this work we generated 5 variant LSTM RNNs. We extracted player paths from four distinct full playthroughs of Super Mario Bros. and trained four individual LSTM RNNs and one combined model. We wished to test if the LSTM RNNs could learn the specific style of the player and incorporate that into generating levels. We incorporated all of the video traces into one final model. Our hypothesis was that a "combined" model could best create levels like the original Super Mario Bros. We hypothesize that the designers created levels to support many different styles, and so a combined representation of multiple styles should better reflect the original game. We provide a brief overview of the four different videos in Table 1. Note that this information is never made available to the generator. Rather, this information is demonstrative of high-level differences between generators.

"Avg. Time" indicates the average amount of time that player spent on a level in that video. "Distance Ratio" represents the average distance traveled by the player relative to the optimal path distance as determined via an automated A* player. "Bonus %" indicates the percentage of "bonus rooms" that player visited. It should be clear from these metrics that each video represents a distinct style of play.

We chose to compare the output of our generators to determine if they encoded player style. To perform this evaluation we followed Summerville and Mateas and used each generator to generated 4000 levels, 2000 with their "above-ground" seed and 2000 with their "below-ground" seed. For comparison we used the previously generated levels from (Summerville and Mateas 2016) found at http://tinyurl.com/SMBRNN (**SPD** in the table) and the original Super Mario Bros. levels (**SMB** in the table). With the **SPD** levels we can compare the results of generating levels with a generator trained on simulated player paths to one trained on human player paths, and with the **SMB** levels we can compare our levels to those designed by a human expert.

We ran two distinct evaluations across these seven sets of levels. For the first we made use of 7 of the designer-focused metrics (Horn et al. 2014; Canossa and Smith 2015) from Summerville and Mateas: l (the leniency, an approximation of difficulty), R^2 (the linearity, how closely the level matches a straight line), i (percentage of objects Mario can interact with), e (percentage of empty space), n (percentage of negative space), p (percentage of level taken up by optimal path), j (# of jumps), and j_i (# of induced jumps). These metrics allow us to determine in what ways, if any, the generated levels differ from one another structurally. We used

	l	R^2	i	e	n	p	j	j_i
K-W	α	β	α	α	β	α	β	β
M-W		C	D	D	SMB	C, D	A	SMB

Table 2: For each metric whether the Kruskal-Wallis (K-W) test found a significant difference between the generators ($\alpha \Rightarrow p < 1e{-}10$, $\beta \Rightarrow p < 1e{-}5$), and for each metric which generators were found to be significantly different from the Mann-Whitney test (M-W).

Figure 4: Example output generator A.

the Kruskal-Wallis test to determine if any of the generators differed from the rest in a statistically significant way for each of these metrics. We also used the Mann-Whitney test to determine which of the generators were significantly different from the rest, using a Bonferroni correction given the post-hoc nature of the analysis. Both the Kruskal-Wallis and Mann-Whitney are non-parametric tests that determine if two samples come from different distributions but make no underlying assumptions about the distributions (unlike Student's t-test which assumes a Normal distribution).

The second of these two evaluations made use of Guzdial and Riedl's learned Super Mario Bros. models (Guzdial and Riedl 2016). These models have been shown to score levels in a way that correlates strongly with human players' perception of level style. Therefore this evaluation allows us to determine if the levels differ from the original Super Mario Bros. level style. For further information on these models please see (Guzdial and Riedl 2016).

Design Metrics Results

We calculated the seven metrics for each of the 4000 output levels from each generator, with the results of this evaluation summarized in Table 2. With the Kruskal-Wallis test we were able to reject the null hypothesis that all of the generators came from the same population (p<1e-5). We further confirmed that for each metric except leniency, one generator differed significantly from all the others via the Mann-Whitney U test. We represent in bold the median values of the distributions that differed significantly lower or higher than all the others (p<0.01). Even though for leniency (l) there was no one generator ranked higher or lower than the rest, the Kruskal-Wallis test rejected the null hypothesis. We visualize the distributions of values for each metric in Figure 5.

To better illustrate the generator's success in learning player style we address each of the individual video generators, referring back to the metrics in Table 1:

A) Player A visited nearly every single bonus room. Given that the generators do not generate bonus areas, this appears as "teleporting" to them (the player path disappearing and reappearing later in a generated level). While none of the metrics capture this, we see that the output

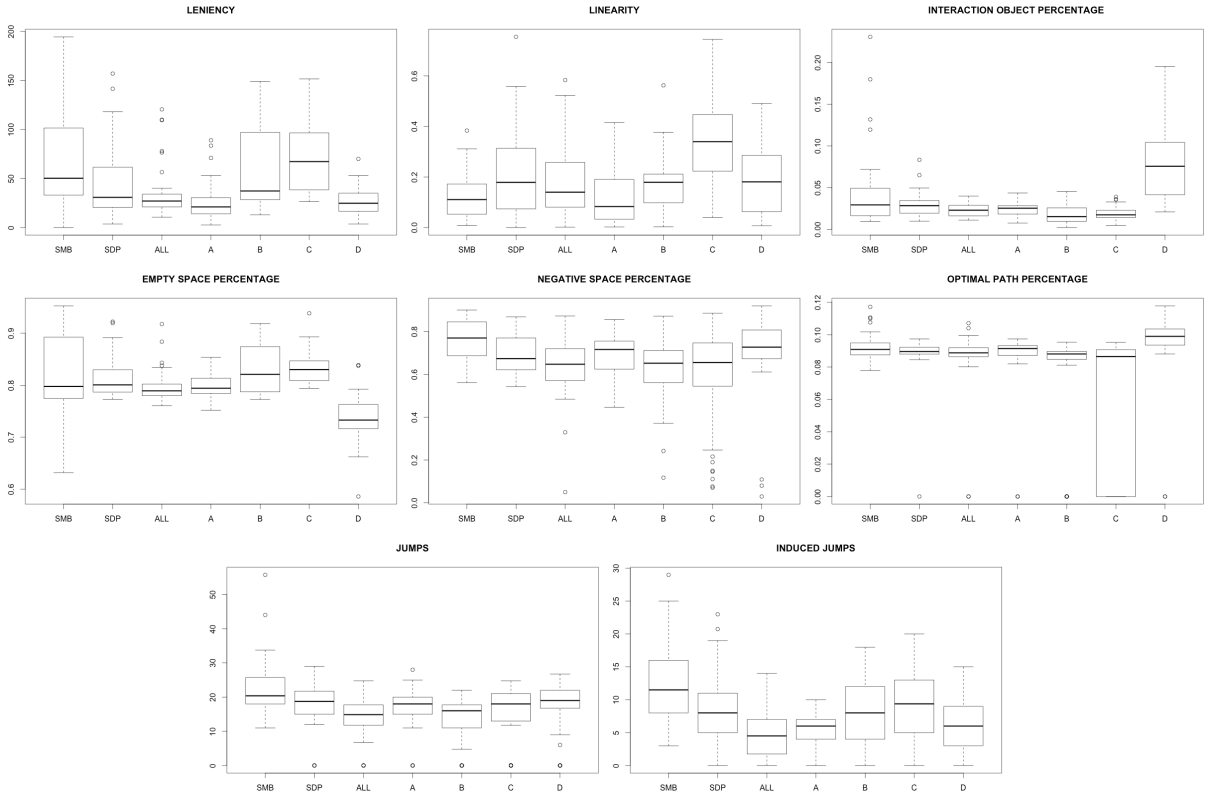

Figure 5: Boxplots for all of the metric evaluations.

of generator A incorporates more teleportation than any other generator as evidenced by Figures 4 and 6.

B) Player B was very much "in the middle" for all the metrics we captured in Table 1, similarly none of its distributions dominated any others. We cannot point to a set of characteristics that make this generator unique, but given how "average" player B was, we argue this is actually a positive sign.

C) Player C sped through each level, and went into no bonus areas. Generator C, as can be seen in Table 2 and Figure 5 has significantly lower p, and significantly higher R^2 than any other generator, indicating it produces flatter levels, where the optimal path requires little exploration.

D) Player D traveled the furthest average distance of the four players. From the video we note this is due to the player going out of the way to collect items and destroy enemies. In Table 2, generator D had significantly higher i (interaction percentage), significantly lower e (empty space percentage), and significantly higher p (percentage of levels taken up by optimal path). This indicates that generator D creates levels that allow for more interaction and require more movement.

The differences in each generator provide strong evidence that the generators learned to reflect individual play style.

	SPD	A	B	C	D	ALL
p-value	0.66	**6e-3**	**2.8e-3**	0.03	**1e-5**	0.3

Table 3: p-values for Mann-Whitney U test between each generator and SMB in terms of style metric.

Style Model Results

We summarize the results of the style metric evaluation in Table 3. For this evaluation, we use a scoring metric that captures the average probability of level components, and has been shown to correlate strongly with human subject's level "style" rankings (Guzdial and Riedl 2016). We score each level from each generator, giving us a distribution of style scores. We then compare each generator's distribution to the original Super Mario Bros. Note that these Super Mario Bros. levels were the same used for the play trace extraction, thus they did not include bonus rooms. We report the outcome of this test in Table 3 with significant values in bold ($p<0.01$). In other words, bold p values suggest that, that generator differs stylistically from Super Mario Bros..

From the table we see that all Video generators, except for Video C, differ significantly in terms of this style metric. However, the combined generator of all videos did not differ significantly. This offers support to our hypothesis that a combination of different play styles would better match the original Super Mario Bros. levels, specifically in terms

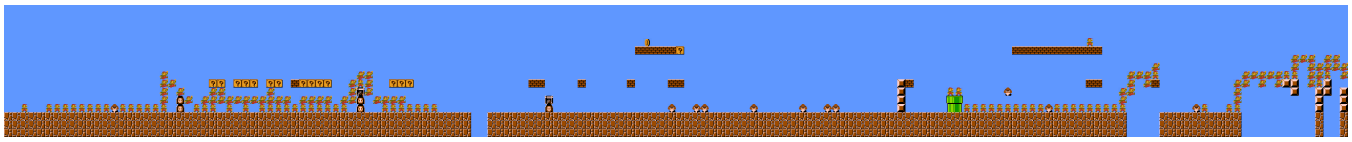

Figure 6: Above Ground Level 1223 from Video A

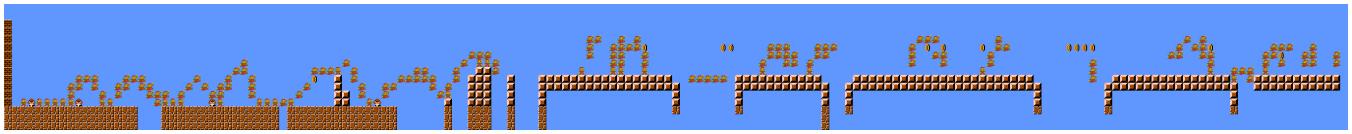

Figure 7: Under Ground Level 689 from Video B

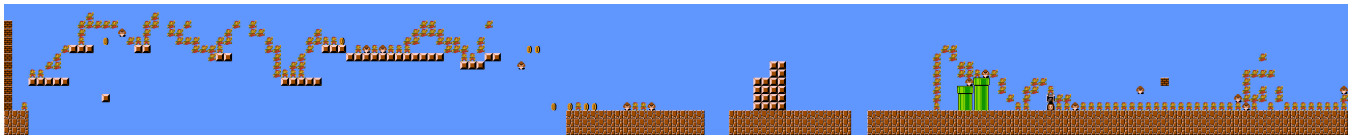

Figure 8: Under Ground Level 143 from Video C

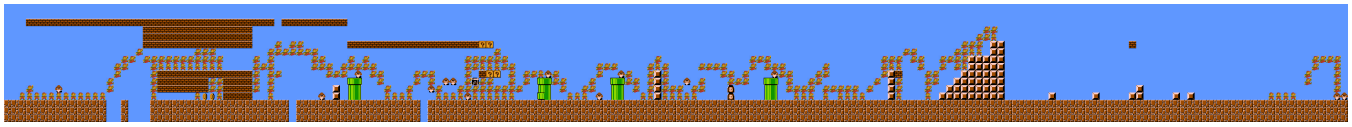

Figure 9: Above Ground Level 530 from Video D

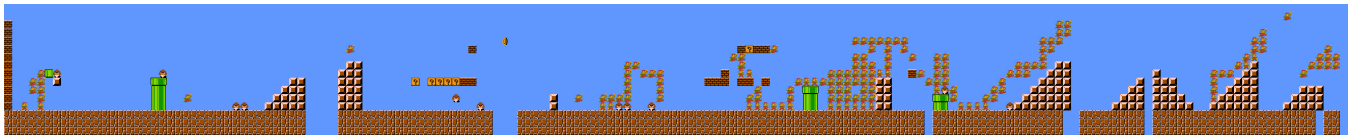

Figure 10: Under Ground Level 488 from All Videos

of level design style. Generators C and SPD, the artificial player, did not differ significantly from the original Super Mario Bros. levels. For SPD we contend this is due to the artificial agent taking near-optimal paths through each level, therefore not biasing the generator to increase non-essential elements from what is found in the original levels. As can be seen in Table 1, Video C is the most like the artificial agent of any human players, taking the least amount of time, with the minimal exploration, and without going to any "bonus areas".

Example Output

In this section we present a set of randomly sampled levels from each of the video generators, seen in Figure 10. To see each of the 4,000 levels generated for each generator, go to `http://tinyurl.com/SMB-from-Video`. While some of these generators demonstrate a less than perfect understanding of Mario's mechanics, overall they output playable, interesting levels.

Conclusions and Future Work

We are altogether pleased with the performance of our generators, however there are a number of features of the original Super Mario Bros. they do not encode. We previously mentioned that they do not handle any level elements that affect Mario's movement (e.g. springs and moving platforms), but they also currently do not represent any "decorative" elements. We look to extend the generators in future work. Also, though our process is successfully statistically biasing levels based on implicit play style, a further study would be needed to determine the effect of this biasing on a player's subjective experience.

Experience driven procedural content generation research has to this point been hampered by the need to collect player data via in-person experiments or from some corpus. With this paper we demonstrate a vast alternative source for this work and demonstrate its viability. In this paper we discuss an approach to encode play style in level generators, via extracting player paths from video. We demonstrated the process for extracting the path from video and how this infor-

mation feeds into an LSTM RNN. Through two evaluations on the output of these generators we demonstrated both that the generated levels differed significantly from each other, and that the generated levels appeared to match the style of the player whose video each was trained on. Taken together this represents a system capable of biasing generated output to match any potential player, only requiring video of that player playing the game.

References

Barbosa Jacob, L.; Kohwalter, T. C.; Clua, E. W.; De Oliveira, D.; and Machado, A. F. 2014. A Non-intrusive Approach for 2D Platform Game Design Analysis Based on Provenance Data Extracted from Game Streaming. In *Computer Games and Digital Entertainment (SBGAMES), 2014 Brazilian Symposium on*, 41–50. IEEE.

Canossa, A., and Smith, G. 2015. Towards a Procedural Evaluation Technique: Metrics for Level Design. *Proceedings of the Tenth International Conference on Foundations of Digital Games* 7:8.

Dahlskog, S., and Togelius, J. 2014. A Multi-level Level Generator. In *2014 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Gers, F. A.; Schmidhuber, J.; and Cummins, F. 2000. Learning to forget: Continual prediction with LSTM. *Neural computation* 12(10):2451–2471.

Guzdial, M., and Riedl, M. 2016. Learning to Blend Computer Game Levels. In *Proceedings of the Seventh International Conference on Computational Creativity*.

Hastings, E. J.; Guha, R. K.; and Stanley, K. O. 2009. Evolving Content in the Galactic Arms Race Video Game. In *2009 IEEE Symposium on Computational Intelligence and Games*, 241–248. IEEE.

Hendrikx, M.; Meijer, S.; Velden, J. V. D.; and Iosup, A. 2013. Procedural Content Generation for Games: A Survey. *ACM Trans. Graph.* 9(1):1:1–1:22.

Hochreiter, S., and Schmidhuber, J. 1997. LSTM can Solve Hard Long Time Lag Problems. *Advances in Neural Information Processing Systems* 473–479.

Hoover, A. K.; Cachia, W.; Liapis, A.; and Yannakakis, G. N. 2015. Audioinspace: Exploring the creative fusion of generative audio, visuals and gameplay. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*. Springer. 101–112.

Hoover, A. K.; Togelius, J.; and Yannakis, G. N. 2015. Composing Video Game Levels with Music Metaphors through Functional Scaffolding. In *Proceedings of the First ICCC Workshop on Computational Creativity and Games*.

Horn, B.; Dahlskog, S.; Shaker, N.; Smith, G.; and Togelius, J. 2014. A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework. *Proceedings of the Ninth International Conference on Foundations of Digital Games*.

Hsieh, J.-L., and Sun, C.-T. 2008. Building a player strategy model by analyzing replays of real-time strategy games. In *2008 IEEE International Joint Conference on Neural Networks*, 3106–3111. IEEE.

Jain, R.; Isaksen, A.; Holmgård, C.; and Togelius, J. 2016. Autoencoders for Level Generation, Repair, and Recognition.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level Control through Deep Reinforcement Learning. *Nature* 518(7540):529–533.

Pulli, K.; Baksheev, A.; Kornyakov, K.; and Eruhimov, V. 2012. Real-time Computer Vision with OpenCV. *Commun. ACM* 55(6):61–69.

Shaker, N.; Togelius, J.; Yannakakis, G. N.; Weber, B.; Shimizu, T.; Hashiyama, T.; Sorenson, N.; Pasquier, P.; Mawhorter, P.; Takahashi, G.; et al. 2011. The 2010 Mario AI championship: Level generation track. *Computational Intelligence and AI in Games, IEEE Transactions on* 3(4):332–347.

Shaker, N.; Shaker, M.; and Abou-Zleikha, M. 2015. Towards Generic Models of Player Experience. In *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press.

Snodgrass, S., and Ontañón, S. 2014. Experiments in Map Generation using Markov Chains. In *Proceedings of the Ninth International Conference on Foundations of Digital Games*.

Stanley, K. O.; Bryant, B. D.; and Miikkulainen, R. 2005. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation* 9(6):653–668.

Summerville, A., and Mateas, M. 2016. Super Mario as a String: Platformer Level Generation Via LSTMs. In *Proceedings of the First International Conference of DiGRA and FDG*.

Togelius, J.; De Nardi, R.; and Lucas, S. M. 2007. Towards Automatic Personalised Content Creation for Racing Games. In *2007 IEEE Symposium on Computational Intelligence and Games*, 252–259. IEEE.

Yannakakis, G. N., and Togelius, J. 2011. Experience-driven Procedural Content Generation. *Affective Computing, IEEE Transactions on* 2(3):147–161.

Yannakakis, G. N.; Spronck, P.; Loiacono, D.; and André, E. 2013. Player Modeling. In Lucas, S. M.; Mateas, M.; Preuss, M.; Spronck, P.; and Togelius, J., eds., *Artificial and Computational Intelligence in Games*, volume 6 of *Dagstuhl Follow-Ups*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 45–59.